Counting Linear Extensions in Practice : MCMC versus Exponential Monte Carlo

Talvitie, Topi Laurinpoika

AAAI Press
2018

Talvitie , T L , Kangas , J-K W , Niinimäki , T M & Koivisto , M K H 2018 , Counting Linear Extensions in Practice : MCMC versus Exponential Monte Carlo . in Thirty-Second AAAI Conference on Artificial Intelligence . Proceedings of the AAAI Conference on Artificial Intelligence , AAAI Press , Palo Alto, CA , pp. 1431-1438 , 32nd AAAI Conference on Artificial Intelligence , New Orleans , United States , 02/02/2018 . < https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16957 >

http://hdl.handle.net/10138/298469

Counting Linear Extensions in Practice: MCMC versus Exponential Monte Carlo

Topi Talvitie
Dept. of Computer Science
University of Helsinki
topi.talvitie@helsinki.fi

Kustaa Kangas
Dept. of Computer Science
Aalto University
juho-kustaa.kangas@aalto.fi

Teppo Niinimäki
Dept. of Computer Science
Aalto University
teppo.niinimaki@aalto.fi

Mikko Koivisto
Dept. of Computer Science
University of Helsinki
mikko.koivisto@helsinki.fi

Abstract

Counting the linear extensions of a given partial order is a #P-complete problem that arises in numerous applications. For polynomial-time approximation, several Markov chain Monte Carlo schemes have been proposed; however, little is known of their efficiency in practice. This work presents an empirical evaluation of the state-of-the-art schemes and investigates a number of ideas to enhance their performance. In addition, we introduce a novel approximation scheme, adaptive relaxation Monte Carlo (ARMC), that leverages exact exponential-time counting algorithms. We show that approximate counting is feasible up to a few hundred elements on various classes of partial orders, and within this range ARMC typically outperforms the other schemes.

1 Introduction

A partially ordered set or *poset* is a set of elements coupled with a binary relation that defines the mutual order between some pairs of elements while leaving other pairs incomparable. A fundamental property of a poset is the set of its linear extensions, the possible ways to extend the poset into a linear order, where all pairs of elements are comparable. This paper concerns the problem of counting the linear extensions of a given poset, which arises in various applications such as sorting (Peczarski 2004), sequence analysis (Mannila and Meek 2000), convex rank tests (Morton et al. 2009), preference reasoning (Lukasiewicz, Martinez, and Simari 2014), partial order plans (Muise, Beck, and McIlraith 2016), and learning graphical models (Wallace, Korb, and Dai 1996; Niinimäki, Parviainen, and Koivisto 2016).

Computing the number of linear extensions of a given poset is #P-complete (Brightwell and Winkler 1991) and thus likely to be intractable in the general case. The fastest known algorithms are based on dynamic programming and require exponential time and space (De Loof, De Meyer, and De Baets 2006; Kangas et al. 2016). Polynomial-time algorithms are known for certain notable special cases such as when the poset is series-parallel (Möhring 1989), when its Hasse diagram is a tree (Atkinson 1990), or when a certain structural parameter is bounded, such as the width (De Loof, De Meyer, and De Baets 2006), the treewidth of the Hasse diagram (Kangas et al. 2016), or the treewidth of the incomparability graph (Eiben et al. 2016).

If an approximation with given probability is sufficient, the problem admits a polynomial-time solution for all posets using Markov chain Monte Carlo (MCMC) methods. The first fully-polynomial time approximation scheme of this kind for was based on algorithms for approximating the volumes of convex bodies (Dyer, Frieze, and Kannan 1991). The later improvements were based on rapidly mixing Markov chains in the set of linear extensions (Karzanov and Khachiyan 1991; Bubley and Dyer 1999) combined with Monte Carlo counting schemes (Brightwell and Winkler 1991; Banks et al. 2017). The mixing times of the Markov chains were recently studied empirically by Talvitie, Niinimäki and Koivisto (2017) and found to be often much better than the worst-case bounds. However, when approximation guarantees are desired, the degrees of the polynomial worst-case bounds can be prohibitively high. It is thus currently unclear to what extent the MCMC-based schemes are usable in practical applications, especially in comparison to the recent improvements in exact counting.

In this work we investigate the feasibility of approximate counting of linear extensions in practice by presenting what is to our knowledge the first empirical evaluation of the state-of-the-art approximation schemes. In addition to this empirical review, we present two algorithmic contributions: First, we propose practical amendments to improve the performance of certain MCMC-based schemes. Second, we extend the exact exponential-time algorithms to approximate counting via a suitable relaxation of the problem, combined with a Monte Carlo approach. As a result, we obtain a novel approximation scheme, adaptive relaxation Monte Carlo (ARMC). Through the empirical evaluation, we show that approximate counting is feasible on posets up to a few hundred elements. We also demonstrate that within this feasible region our ARMC scheme vastly outperforms the other approximation schemes.

We introduce some central concepts of partially ordered sets in Section 2. In Section 3 we review the state-of-the-art MCMC schemes and present our practical improvements. In Section 4 we review the exact exponential-time algorithms and present our ARMC scheme that leverages them for approximate counting. Section 5 presents an empirical evaluation of the schemes described in Sections 3 and 4.

2 Preliminaries

A *partial order* $\prec$ is an irreflexive and transitive binary relation. A *partially ordered set* or *poset* is a pair $P = (A, \prec)$, where A is a set and $\prec$ is a partial order on A. A pair $(x, y) \in \prec$ is called an *(ordering) constraint* and denoted $x \prec y$ for short; we say that x is a *predecessor* of y and y is a *successor* of x. A pair of elements $x, y \in A$ are *comparable* if $x \prec y$ or $y \prec x$; otherwise they are *incomparable*. If all pairs of elements are comparable, we say that P is a *linear order*. We visualize a poset as a directed acyclic graph with A as the vertex set and $\prec$ as the arc set (Fig. 1). For all $x \prec y \prec z$, the arc $x \to z$ is implied by transitivity and can be omitted for clarity, producing the *cover graph* of P.

A poset $P' = (A, \prec')$ is an *extension* of $P = (A, \prec)$ if $x \prec y$ implies $x \prec' y$ for all $x, y \in A$; dually, in this case we say that P is a *relaxation* of P'. An extension that is also a linear order is called a *linear extension*. We denote by $\mathcal{L}(P)$ the set of linear extensions and by $\ell(P) = |\mathcal{L}(P)|$ the number of linear extensions of P. When there is no ambiguity about P, we may extend this notation to any subset $B \subseteq A$, denoting $\ell(B) := \ell(B, \prec_B)$ for short, where $\prec_B$ is the restriction of $\prec$ to the set B.

Throughout the paper, we consider the problem of producing an (ϵ, δ)-approximation of $\ell(P)$: For given $\epsilon, \delta > 0$ and poset P, produce an estimate $\bar{\ell}$ of $\ell(P)$ such that

$$\Pr\big((1+\epsilon)^{-1} < \bar{\ell}/\ell(P) < 1+\epsilon\big) \geq 1 - \delta\,.$$

The approximation schemes considered yield slight variations of this error bound, but they can be parameterized to satisfy the above problem statement.

Reduction Rules

Let $P = (A, \prec)$ be a poset. We attribute to folklore the following equalities that reduce the problem of computing $\ell(P)$ to counting linear extensions in subsets of A.

First, if $\{A_1, A_2\}$ is a partition of A such that no element in A_1 is comparable with an element in A_2, then

$$\ell(P) = \binom{|A|}{|A_1|} \ell(A_1)\ell(A_2)\,. \tag{1}$$

Second, if $\{A_1, A_2\}$ is a partition of A such that $a_1 \prec a_2$ for all $a_1 \in A_1$ and $a_2 \in A_2$, then

$$\ell(P) = \ell(A_1)\ell(A_2)\,. \tag{2}$$

Third, if A is non-empty, then, denoting by $\min P$ the set of elements of P with no predecessors, we have that

$$\ell(P) = \sum_{x \in \min P} \ell(A \setminus \{x\})\,. \tag{3}$$

3 Markov Chain Monte Carlo

The state-of-the-art polynomial-time randomized schemes for approximate counting of linear extensions have been designed from the viewpoint of asymptotic worst-case time complexity. The traditional approach was to build a telescopic product estimator based on self-reducibility and rapidly mixing Markov chains (Brightwell and Winkler 1991; Huber 2006; Bubley and Dyer 1999). A more recent approach, the Tootsie Pop algorithm, is based on random sampling from an embedding of the set of linear extensions into a continuous space (Banks et al. 2017; Huber and Schott 2010). We next review these approaches. For the former approach, we also present a number ideas to enhance the existing schemes when the interest is in practical performance.

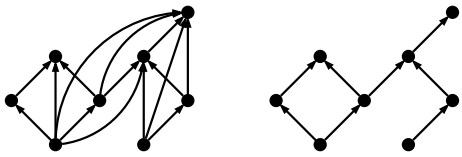

Figure 1: An eight-element poset (left) and its cover graph.

The Telescopic Product Estimator

Brightwell and Winkler (1991) presented the following reduction to (approximately) uniform sampling. Construct a sequence of posets $(P_i)_{i=0}^k$, starting from $P_0 = P$ and ending in a linearly ordered set P_k, such that each P_i is obtained from the previous poset P_{i-1} by adding an appropriately chosen ordering constraint $a_i \prec b_i$ and those that follow by transitivity. Now, write $\ell(P)^{-1}$ as a telescopic product $\prod_{i=1}^k \mu_i$ where $\mu_i = \ell(P_i)/\ell(P_{i-1})$. For each factor μ_i, use the zero–one Monte Carlo estimate $\bar{\mu}_i$ obtained by drawing s independent samples from $\mathcal{L}(P_{i-1})$ (almost) uniformly at random, and computing the proportion of samples that fall in $\mathcal{L}(P_i)$. This yields an estimator $\bar{\mu} = \prod_{i=1}^k \bar{\mu}_i$ whose expected value is $\ell(P)^{-1}$. The smaller the factors μ_i are, the higher the number of samples s per factor has to be to estimate them up to sufficient relative error. Supposing the factors μ_i are bounded from below by a positive constant, putting $s = O(\epsilon^{-2} k \log \delta^{-1})$ guarantees that $1/\bar{\mu}$ is a $(\epsilon, \delta/2)$-approximation of $\ell(P)$ (Talvitie, Niinimäki, and Koivisto 2017, proof of Thm. 1).

Construction by Sorting. For the reduction to be efficient, Brightwell and Winkler (1991) choose the pairs of elements (a_i, b_i) such that k is small while ensuring that the factors μ_i are bounded from below by a positive constant. This is achieved by simulating any $O(n \log n)$-time comparison sorting algorithm on the set of elements A, iteratively constructing the sequence $(P_i)_{i=0}^k$ starting from the original poset P. For each comparison between a pair of elements (x, y) made by the algorithm, we look up the ordering from the currently last element P_i in the sequence. If the elements are incomparable, then we use the Monte Carlo method with a sufficient number of samples to find out which order between the elements is more probable in the linear extensions of P_i, and add P_i augmented with that ordering constraint to the end of the sequence, i.e., (a_{i+1}, b_{i+1}) is either (x, y) or (y, x). An analysis shows that in total $O(\epsilon^{-2} n^2 \log^2 n \log \delta^{-1})$ samples from the uniform distribution of linear extensions (of the varying posets) suffice for an (ϵ, δ)-approximation of $\ell(P)$ (Talvitie, Niinimäki, and Koivisto 2017); if the samples are from an *almost* uniform

distribution, then the bound becomes slightly larger. Using Huber's (2006) perfect sampler that generates a random linear extension in $O(n^3 \log n)$ expected time, the whole algorithm runs in $O(\epsilon^{-2} n^5 \log^3 n \log \delta^{-1})$ expected time.

Structure Decomposition. We next propose a way to improve the performance of the telescopic product scheme. The idea is to take into account the special structure that appears in the posets in the tail of the sequence $(P_i)_{i=0}^k$, provided that we use Quicksort as the sorting algorithm: In the first at most $m = O(n)$ operations, Quicksort chooses a pivot element $p \in A$ and compares all other elements to it. Thus after m operations, p divides the rest of the elements into its predecessors and successors, and we can use the reduction rule (2) to continue the algorithm in the set of predecessors and successors separately. This idea repeats recursively, and using a variant of Quicksort that always uses the median element as the pivot, the structure is divided in half every $O(n)$ comparisons. The time complexity of obtaining a single sample after h halvings is $O(2^h (2^{-h} n)^3 \log(2^{-h} n))$, which decays geometrically as h increases. Thus the time complexity of the $O(n \log n)$ comparisons is dominated by the first $O(n)$ comparisons, which saves a factor of $\log n$ in the running time and yields the best worst-case bound we currently are aware of:

Proposition 1. *There is a randomized algorithm that given $\epsilon, \delta > 0$ and a poset on n elements, computes an (ϵ, δ)-approximation of the number of linear extensions of the poset in $O(\epsilon^{-2} n^5 \log^2 n \log \delta^{-1})$ expected time.*

In practice, we can often significantly expedite the algorithm by decomposing the poset into disjoint parts as soon as it is possible using reduction rules (1) and (2). For example, in the (extreme) case of series–parallel posets the reduction rules are enough to completely decompose the poset without adding new ordering constraints. To avoid the $O(n)$ comparisons required to find the median element, we choose the pivot for Quicksort heuristically based on the current poset instead. The heuristic chooses the element with largest minimum of the number of predecessors and successors in the current poset. This choice of heuristic aims to reduce the number of steps required to decompose the poset into two parts. Theoretically, choosing pivot this way may cause Quicksort to reach its worst case of $O(n^2)$ comparisons, causing a near-linear slowdown in the algorithm, but we found this choice of pivot work well in practice. An example of the algorithm is shown in Fig. 2.

Sampling Linear Extensions. Early proposals of the telescopic product scheme relied on sampling from almost uniform distribution of linear extensions by simulating a suitable Markov chain for given number of iterations. The number of iterations required depends linearly on the mixing time of the Markov chain, for which an upper bound has to be known in advance. For the classical chain of Karzanov and Khachiyan (1991) the mixing time is known to be $\Theta(n^3 \log n)$ (Bubley and Dyer 1999; Wilson 2004), which remains the best known worst-case bound.

Huber's (2006) perfect sampler runs in $O(n^3 \log n)$ expected time and is the fastest known algorithm for sampling exactly from the uniform distribution of linear extensions. It currently yields the best asymptotic running time bound for approximate counting based on the telescopic product scheme. However, Huber's (2014) later perfect sampler, which is based on Gibbs sampling, was recently shown to run significantly faster in various instances of practical size (Talvitie, Niinimäki, and Koivisto 2017), even if the sampler is currently not known to run in polynomial time in the worst case.

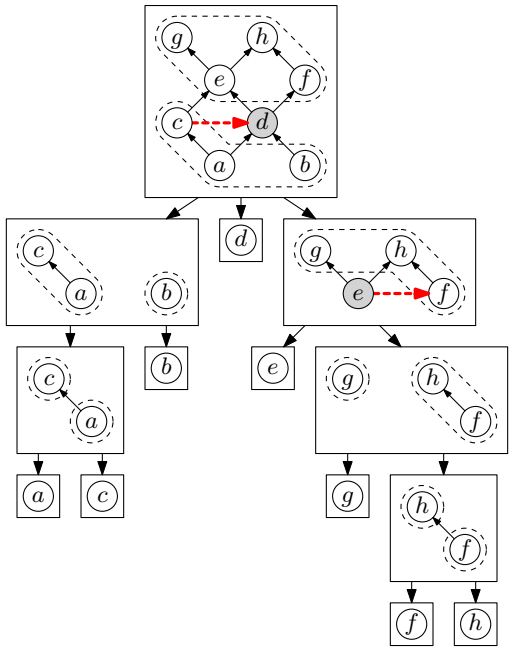

Figure 2: An illustration of structure decomposition, with the proposed improvements. The algorithm chooses the pivot d, and compares it to the only remaining incomparable element c. After adding the ordering constraint $c \prec d$ (probability $\mu_1 = 15/29 \geq 1/2$), the poset is split into three components using the reduction rule (2). One of the resulting components, $\{e, f, g, h\}$, is further split into two components after adding the ordering constraint $e \prec f$ (probability $\mu_2 \geq 3/5$). The remaining subproblems are solved by applying the reduction rules (1) and (2). The number of linear extensions is $\mu_1^{-1} \mu_2^{-1} \binom{3}{2} \binom{3}{1} = \frac{29}{15} \times \frac{5}{3} \times 3 \times 3 = 29$.

The Tootsie Pop Algorithm

Huber and Schott (2010) presented a randomized approximation scheme, called the Tootsie Pop algorithm, for estimating the measure $\mu(\mathcal{A}')$ of a given set $\mathcal{A}'$ in a continuous space under certain conditions. The algorithm estimates a ratio $\mu(\mathcal{A}')/\mu(\mathcal{A}'')$, where $\mathcal{A}''$ is another set whose measure is assumed to be easy to compute, by considering a family of intermediate sets $\{\mathcal{A}_\beta \mid \beta \in \mathbb{R}\}$ such that

- $\mathcal{A}_{\beta_1} \subset \mathcal{A}_{\beta_2}$ for all $\beta_1 < \beta_2$,
- the function $\beta \mapsto \mu(\mathcal{A}_\beta)$ is continuous,
- $\mathcal{A}_{\beta'} = \mathcal{A}'$ and $\mathcal{A}_{\beta''} = \mathcal{A}''$ for some $\beta' > \beta''$,

- there exists an algorithm that, for a given $\beta \in \mathbb{R}$, draws from the uniform distribution on $\mathcal{A}_\beta$.

The algorithm constructs a random sequence $(\beta_i)_{i=0}^{\infty}$ iteratively by setting $\beta_0 = \beta'$ and $\beta_{i+1} = \inf\{\beta \in \mathbb{R} \mid X_i \in \mathcal{A}_\beta\}$, where X_i is drawn uniformly at random from $\mathcal{A}_{\beta_i}$. Now the random variable $Z = \min\{i \geq 0 \mid \beta_{i+1} \leq \beta''\}$ is Poisson distributed with expected value $\log(\mu(\mathcal{A}')/\mu(\mathcal{A}''))$. The expected value is estimated by averaging multiple independent realizations of Z.

Banks et al. (2017) apply the Tootsie Pop algorithm to count linear extensions by embedding the set of linear extensions into a continuous space. Roughly speaking, they define the set $\mathcal{A}_\beta$ as a set of pairs $(P', [0, w'))$, where P' is a linear extension and w' is a distance of P' to an arbitrarily chosen fixed linear extension; the distance is modulated by the parameter β. They also give an algorithm that, for any fixed β, samples exactly from the uniform distribution on the set $\mathcal{A}_\beta$ in $O(n^3 \log n)$ average time, and prove that it yields an (ϵ, δ)-approximation for $\ell(P)$ in $O(\epsilon^{-2} n^3 \log^2 \ell(P) \log n \log \delta^{-1})$ average time.

4 Adaptive Relaxation Monte Carlo

In this section we present a novel scheme for approximate counting of linear extensions, which we dub *adaptive relaxation Monte Carlo* (ARMC). This scheme leverages an exact dynamic programming (DP) approach for counting linear extensions (De Loof, De Meyer, and De Baets 2006), which can be faster than MCMC schemes on posets of moderate size, despite its exponential complexity. We begin by briefly reviewing the DP algorithm and then describe the ARMC scheme to use it in approximate counting.

Exact Dynamic Programming

Given a poset $P = (A, \prec)$, the DP algorithm computes $\ell(P)$ by applying the reduction rule (3) recursively. This computation involves solving the subproblem $\ell(U)$ exactly for each *upset* U of P, a set such that $x \in U$ and $x \prec y$ imply $y \in U$. By caching the subproblem solutions the algorithm can be made to run in $O(|\mathcal{U}|\, w)$ time, where $\mathcal{U}$ is the set of upsets and w is the width of P. Dilworth's theorem (1950) notably implies the bound $|\mathcal{U}| = O(n^w)$ on the number of upsets, though in the worst case we still have $|\mathcal{U}| = 2^n$. Kangas et al. (2016) note further that $\mathcal{U}$ can often be pruned with the reduction rule (1): Whenever through application of rule (3) the cover graph of a subproblem becomes disconnected, rule (1) can be applied to solve the problem independently for each connected component. This modification brings the worst-case running time to $O(2^n n^2)$ for finding the connected components but may in return reduce the number of subproblems exponentially.

Importantly, once the DP algorithm has finished, it also enables straightforward sampling of linear extensions from the uniform distribution: If A is connected, the first element x of the linear extension is drawn from $\min P$, with the probability $p(x) = \ell(A \setminus \{x\})/\ell(A)$, and the order on the remaining elements is sampled by recursing on $A \setminus \{x\}$. If A is not connected, a linear extension is sampled recursively for each connected component and the sampled extensions are then interleaved uniformly at random. In this manner a single sample can be drawn in $O(n^2)$ time.

Monte Carlo via Exact Counting

We now propose a way to apply the DP algorithm to approximate counting. Given a poset P, the basic idea is to relax P by removing ordering constraints until the counting problem becomes feasible for the DP algorithm. We then estimate the error introduced by the removal of constraints using Monte Carlo, similarly to Section 3 where constraints were added instead. Specifically, let R be a relaxation of P, and let $\mu = \ell(P)/\ell(R)$ be the probability that a linear order sampled uniformly from $\mathcal{L}(R)$ is also in $\mathcal{L}(P)$. To approximate $\ell(P)$, we first use the DP algorithm to compute $\ell(R)$ and then sample m linear orders from $\mathcal{L}(R)$ to compute the estimate $\bar{\mu} = \frac{1}{m}\sum_{i=1}^{m} X_i$, where $X_i = 1$ if the ith sample is in $\mathcal{L}(P)$ and $X_i = 0$ otherwise. As m grows, $\bar{\mu}$ concentrates around μ and thus $\bar{\mu}\,\ell(R)$ concentrates around $\ell(P)$. Again, the smaller μ is, the more samples we need for an accurate approximation. The efficiency of this scheme thus depends crucially on how the relaxation R is chosen, as it determines both μ and the time required to run the DP algorithm.

Before discussing the choice of R in detail, consider the number of samples required to reach the desired approximation guarantees. Chebyshev's inequality yields that

$$\Pr(1 - \epsilon \leq \bar{\mu}/\mu \leq 1 + \epsilon) \geq 1 - \delta \tag{4}$$

if we take m to be at least $\lceil (1-\mu)/(\epsilon^2 \delta \mu) \rceil$. However, this quantity depends on the unknown μ and therefore cannot be used directly as a stopping criterion. We use instead an algorithm by Dagum et al. (2000), which adaptively determines when to stop sampling, based on the samples it has seen so far. For any random variable X distributed in $[0, 1]$ with expected value $\mu > 0$ this algorithm requests a number of samples and eventually outputs an approximation $\bar{\mu}$ with the guarantee (4). The number of samples requested is guaranteed to be optimal within a constant factor among all algorithms that achieve the same error bound. In our case we may use a simpler precursor of the general algorithm, which additionally requires that X be a Bernoulli variable. This algorithm stops as soon as it has seen $m_{\epsilon,\delta}$ samples that fall in $\mathcal{L}(P)$, where $m_{\epsilon,\delta}$ only depends on ϵ and δ. Under the practical assumption that both ϵ and δ are bounded from above by a constant, we have that $m_{\epsilon,\delta} = O(\epsilon^{-2} \log \delta^{-1})$.

Partition Relaxations

We now address the crucial step of choosing the relaxation R. The choice of R is effectively a tradeoff between the time spent running the DP algorithm (*DP phase*) and the time spent drawing the samples (*sampling phase*). If R is taken to be close to P, then μ is large and the expected number of samples required until $m_{\epsilon,\delta}$ samples hit $\mathcal{L}(P)$ is small. On the other hand, the more we allow R to deviate from P, the more constraints we can remove to speed up the DP phase. Since finding an optimal balance between the two phases is presumably intractable, we propose an adaptive heuristic strategy for selecting the relaxation.

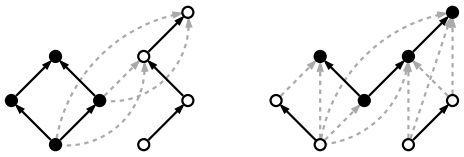

Figure 3: Two relaxations of the poset in Fig. 1, obtained by partitioning the elements into two sets of size $k = 4$ (shown in black and white) and removing all constraints between the sets (dashed arcs). The original poset has 88 linear extensions while the relaxations have 140 and 1260, respectively.

First, consider choosing R so as to restrict the time spent in the DP phase. A natural way to accomplish this is to remove a subset of ordering constraints such that the cover graph is split into multiple connected components. Specifically, we partition A into sets $A_1, A_2, \ldots, A_{\lceil n/k \rceil}$ of size k (the last set may be smaller) and then take $R = (A, \prec')$, where $x \prec' y$ if and only if $x \prec y$ and x and y are in the same set A_i (Fig. 3). To compute $\ell(R)$, the DP algorithm may now immediately apply the reduction rule (1) and thus solves the problem in $O(2^k\, k^2)$ time for each of the $\lceil n/k \rceil$ components. The parameter k controls the tradeoff: increasing k causes the DP phase to take longer, but typically brings R closer to P and thus saves time in the sampling phase.

Assume for now that we are able to choose a good value for k and consider the problem of choosing a partition so that μ is maximized. This is equivalent to minimizing $\ell(R)$, which may vary by orders of magnitude between different R even for a fixed value of k, as illustrated in Fig. 3. Aiming to minimize $\ell(R)$, we employ a search in the space of possible partitions. Since computing $\ell(R)$ is still relatively expensive, we require a more computationally feasible function for comparing candidate partitions. We choose the number of ordering constraints removed in the relaxation as the function to minimize, since removing a constraint from a poset always increases the number of linear extensions. While the number of constraints induced by a partition is easy to compute, minimizing the number subsumes the minimum bisection problem, which is NP-complete (Garey, Johnson, and Stockmeyer 1976). Instead of attempting an optimal solution, we resort to greedy hillclimbing that starts at a random partition and iteratively swaps pairs of elements between different A_i that yield the greatest decrease in the number of removed constraints. We find that this strategy works well in practice, even if it may not converge to a global optimum.

Adaptive Balancing

To complete the ARMC scheme, we design an adaptive strategy for choosing the parameter k. While various strategies could be used for different types of posets, our focus here is on an "average" poset of relatively uniform (random) structure and varying width and density. For such posets, a typical empirical trend is that increasing k will exponentially increase the time spent in the DP phase while decreasing k may exponentially increase the time spent in the sampling phase. Figure 4 illustrates this behavior on two posets used in the experiments in Section 5. We rely on the empirical observation that a good choice of k for minimizing the total running time tends to be where the two phases take roughly the same amount of time. To find such a point, we propose the following procedure that starts at a low value of k and iteratively increases it until the phases are in balance:

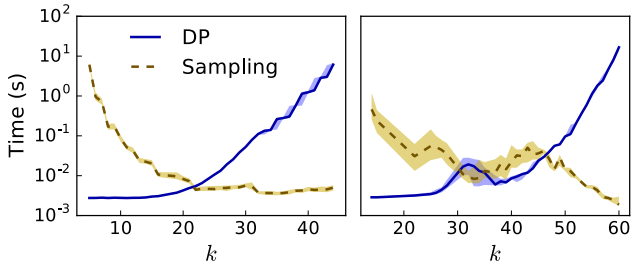

Figure 4: The time spent in the DP phase and the sampling phase as a function of k on two randomly generated posets on 64 elements, a bipartite poset (left) and a poset of average degree 5 (right). The lines show the median and the quartiles over 99 runs. The variance is due to hillclimbing starting at random partitions and converging to different relaxations.

1. Choose a partition relaxation R with the parameter k.
2. Run the DP algorithm on R to obtain $\ell(R)$ and to enable efficient sampling from $\mathcal{L}(R)$.
3. Try running the sampling phase for R tentatively, but stop sampling after time $\alpha\, T_2$, where T_2 is the time spent in step 2 and $\alpha \leq 1$ is a constant.
4. Let $t_3 = m_{\epsilon,\delta}\, T_3/m$ be estimated time to run the sampling phase in full, where T_3 is the time spent sampling in step 3, and m is the number of samples that fell in $\mathcal{L}(P)$.
5. Let $t_2 = \beta\, T_2$ be estimated time to run the DP algorithm for a larger k, where $\beta > 1$ is a constant.
6. If $t_3 < t_2$, accept R and run the sampling phase in full, otherwise increase k by a constant κ and go to step 1.

The procedure increases k until the sampling phase is empirically estimated to run at least as fast as the DP phase. By the choice of α, the time spent in step 2 bounds the time spent in step 3. Since the time required by the other steps is negligible by comparison and the time required in step 2 increases exponentially in k, the final iteration of step 2 is likely to dominate the total running time. Combining this time with the expected time spent in the sampling phase, we obtain the following rough characterization.

Proposition 2. *The expected running time of the ARMC scheme is* $O(2^k\, k\, n + n^2 \mu^{-1} \epsilon^{-2} \log \delta^{-1})$.

The first term follows from the analysis of the DP phase, while $\mu^{-1}\epsilon^{-2}\log\delta^{-1}$ is the expected number of samples required and n^2 is the time spent in drawing each sample. The latter factor is in practice much smaller as often the first few elements already show that the sample is not in $\mathcal{L}(P)$.

In the experiments we set $\alpha = 0.1$, $\beta = 10$, and $\kappa = 5$, observed to perform well on average. As a practical optimization, we also keep the previous relaxation found in step 2 if it has fewer linear extensions than the new relaxation. Further, if k becomes so large that the DP algorithm starts running out of memory, we proceed directly to the sampling phase with the best relaxation found so far.

While the adaptive strategy enjoys empirical success, it can perform poorly in certain extreme cases. In particular, it fails to quickly detect cases where relaxations offer no benefit over the exact DP algorithm (posets of very low width). and $k = n$ is thus best choice. Very skewed structures may also cause t_2 to be underestimated and thus k to be increased too much. Depending on the application, these flaws can be partially remedied by employing hybrid solutions that apply specialized algorithms for edge cases and ARMC otherwise.

5 Experiments

We empirically evaluated the following MCMC schemes and the ARMC scheme for counting linear extensions described in the previous sections:

Telescopic Product. The basic scheme due to Brightwell and Winkler (1991), revisited by Talvitie et al. (2017).

Decomposition Telescopic Product. Like the Telescopic Product scheme but with structure decomposition.

Decomposition Telescopic Product w/ Gibbs. Like the Decomposition Telescopic Product scheme, but replacing Huber's (2006) $O(n^3 \log n)$ average time linear extension sampler by Huber's (2014) newer Gibbs sampler.

Tootsie Pop. The scheme due to Banks et al. (2017).

Exact Dynamic Programming. The algorithm obtained by combining the reduction rules (3) and (1) as presented by Kangas et al. (2016).

Adaptive Relaxation Monte Carlo. The scheme presented in Section 4.

We evaluated the schemes on instances of different sizes generated from the following classes of posets:

- AvgDeg(k) for $k \in \{3, 5\}$: The poset is generated as the transitive closure of a directed acyclic graph with vertices $\{1, 2, \ldots, n\}$ with expected average degree k generated by adding each edge (i, j) where $1 \leq i < j \leq n$ with probability $k/(n-1)$.
- Bipartite(p) for $p \in \{0.2, 0.5\}$: Poset where the set of elements are split into two equally sized parts X and Y, and an ordering constraint $x \prec y$ is added for each pair $(x, y) \in X \times Y$ with probability p.
- Posets extracted from benchmark Bayesian networks, obtained from the Bayesian Network Repository (`www.cs.huji.ac.il/~galel/Repository`). We consider five networks of varying structure: Andes, Diabetes, Link, Munin, and Pigs. Each poset is generated as the transitive closure of a random subgraph of a base network. The subgraph is induced from a subset of nodes, chosen by starting at a random node and iteratively adding random nodes that already have at least one neighbor in the set.

We generated five posets from each instance class and size between 8 and 512 elements. We will make these instances as well as all algorithm implementations publicly available.[1]

We ran each algorithm on each instance in single thread. Each run was limited to 24 hours of CPU time and 8 GB of RAM; all running times were measured. Each algorithm was instantiated to produce a $(1, 1/4)$-approximation for the number of linear extensions, i.e., an estimate within a factor of 2 with probability at least $3/4$. Note that this particular choice of the parameters ϵ and δ enables extrapolation also to other pairs of values: for all algorithms, except for exact dynamic programming, the running times are roughly multiplied by $\epsilon^{-2} \log_4 \delta^{-1}$. Figure 5 summarizes the results.

The results are generally very similar on all instance classes. The Tootsie Pop algorithm is faster than the basic telescopic product scheme, but the structure decomposition optimization improves the performance significantly, enough to make it faster than Tootsie Pop in most cases. The curves for the MCMC algorithms resemble straight lines in the logscale plots, which means that their growth is roughly polynomial. Both the structure decomposition optimization and the Gibbs sampler reduce the slope of the curve, both effectively improving the performance by 1–2 orders of magnitude in the largest cases solved by the MCMC algorithms.

Within our time limit, adaptive relaxation Monte Carlo is clearly the fastest: it solves in seconds instances that take hours with MCMC. However, by extrapolating the curves it seems likely that the fastest MCMC scheme will overtake the exponential algorithms if the computations are allowed to take weeks. In small instances, even the exact dynamic programming algorithm is very fast compared to the approximate MCMC algorithms.

6 Conclusions

We have investigated the problem of approximate counting of linear extensions from a practical perspective. We presented several ideas to enhance state-of-the-art MCMC schemes, and as a side result, improved the best known worst-case bound (Proposition 1). We also introduced a novel scheme, adaptive relaxation Monte Carlo (ARMC), which exploits the fact that small instances can often be handled very fast by dynamic programming (Kangas et al. 2016). Our empirical evaluation on various instances that can be solved in matter of hours showed that, whereas the enhancements expedite the MCMC schemes by 1–2 orders of magnitude, ARMC is multiple orders of magnitude faster. Altogether, the presented improvements are expected to translate to more efficient applications where counting linear extensions is a crucial step, such as in learning Bayesian networks (Niinimäki, Parviainen, and Koivisto 2016).

It is worth noting that our study only concerned serial computation using a single thread. Unlike the ARMC scheme, the MCMC schemes allow straightforward parallelization to multiple multi-core machines; a related advantage is that they require relatively little (i.e. polynomial) space. Thus in some circumstances, the enhanced MCMC scheme may be the best option for approximate counting.

[1] `github.com/ttalvitie/le-counting-practice`

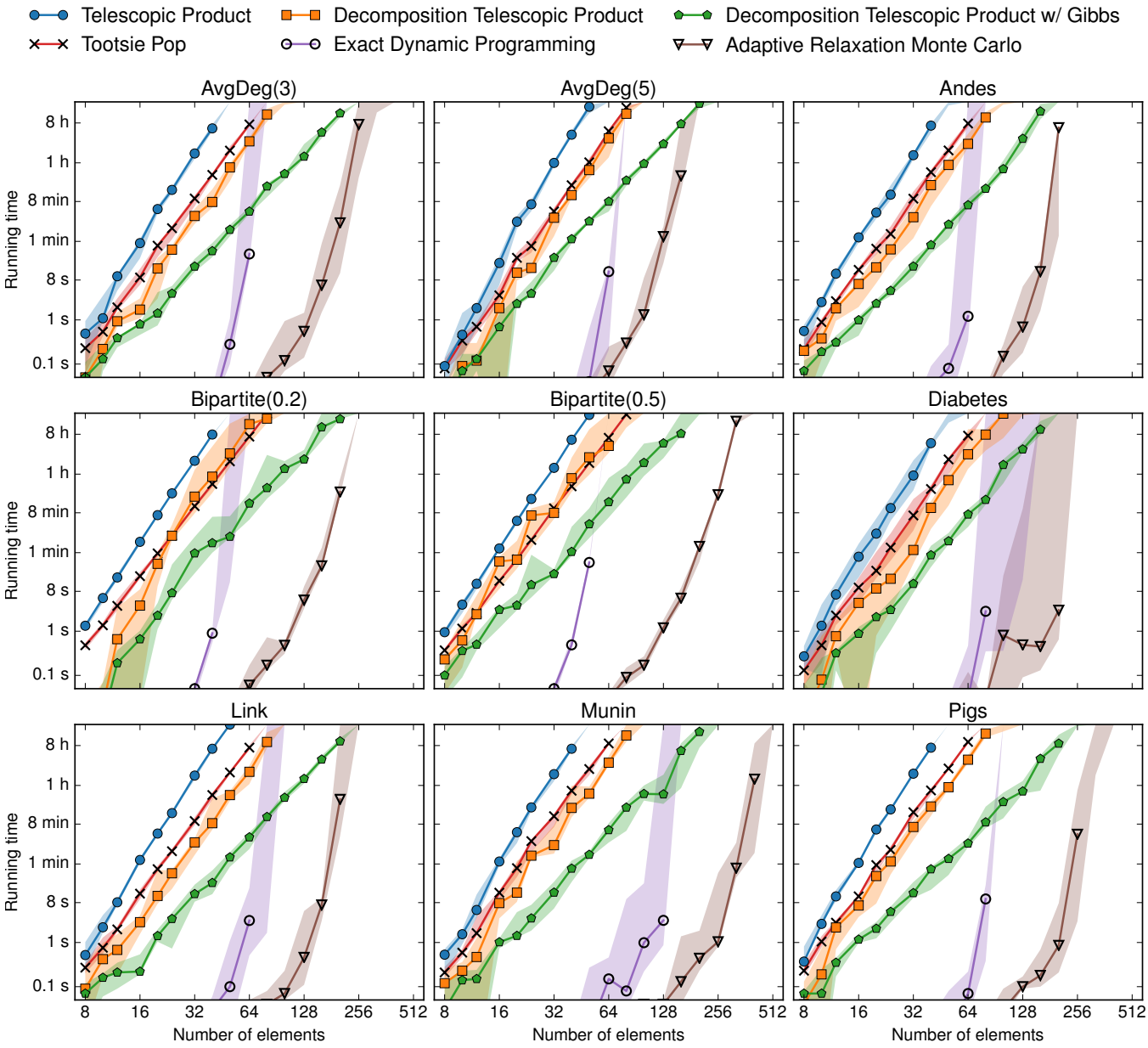

Figure 5: The running times of approximation algorithms for counting linear extensions on different classes of posets as functions of the number of elements in the poset. All algorithms estimate the count within a factor of 2 with probability at least $3/4$. The markers show the median of five independent runs, and the surrounding shaded area shows the range of the values. The time limit of 24 hours is always the limiting factor for all other algorithms except for exact dynamic programming, which always reaches the memory limit first. Both axes are logarithmic, which means that polynomials should appear as straight lines. Very small running times are outside the visible area.

We believe our instantiation of the ARMC scheme leaves room for further improvements. The key step in the scheme is the choice of the relaxation R, which determines the time required by the two main phases: computing $\ell(R)$ and sampling from $\mathcal{L}(R)$. The present work employed a simple local search in a subspace of relaxations, but we surmise that a more intelligent search strategy could improve the choice substantially. Further, to compute $\ell(R)$ we relied on the exponential dynamic programming algorithm, which is applicable to all posets. Alternatively, one could choose a special R that admits polynomial-time counting and sampling, such as a tree or a series–parallel poset; though our tentative experiments suggest such relaxations tend to require impractically many samples, a more careful analysis is warranted.

Our results on counting linear extensions raise questions about other intractable counting problems. On one hand,

can existing MCMC schemes for which known worst-case bounds are impractical be enhanced by ideas similar to the present work? On the other hand, do other problems admit practical exponential-time schemes analogous to ARMC?

Finally, we note that there has been recent progress in alternative paradigms for approximate counting, which leverage the power of modern SAT and ILP solvers for hard decision and optimization problems (Gomes, Sabharwal, and Selman 2006; Chakraborty, Meel, and Vardi 2014; Chakraborty et al. 2015; Kim, Sabharwal, and Ermon 2016). We find it as an intriguing open question, whether such tools can be successfully applied to counting linear extensions.

Acknowledgments

This work was supported in part by the Academy of Finland, under Grants 276864, 303816, and 284642 (Finnish Centre of Excellence in Computational Inference Research COIN).

References

Atkinson, M. D. 1990. On computing the number of linear extensions of a tree. *Order* 7(1):23–25.

Banks, J.; Garrabrant, S.; Huber, M. L.; and Perizzolo, A. 2017. Using TPA to count linear extensions. *arXiv preprint arXiv:1010.4981*.

Brightwell, G., and Winkler, P. 1991. Counting linear extensions. *Order* 8(3):225–242.

Bubley, R., and Dyer, M. 1999. Faster random generation of linear extensions. *Discrete Mathematics* 201(13):81–88.

Chakraborty, S.; Fried, D.; Meel, K. S.; and Vardi, M. Y. 2015. From weighted to unweighted model counting. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, 689–695. AAAI Press.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2014. Balancing scalability and uniformity in SAT witness generator. In *Proceedings of the 51st Annual Design Automation Conference*, 60:1–60:6.

Dagum, P.; Karp, R. M.; Luby, M.; and Ross, S. M. 2000. An optimal algorithm for Monte Carlo estimation. *SIAM Journal on Computing* 29(5):1484–1496.

De Loof, K.; De Meyer, H.; and De Baets, B. 2006. Exploiting the lattice of ideals representation of a poset. *Fundamenta Informaticae* 71(2,3):309–321.

Dilworth, R. P. 1950. A decomposition theorem for partially ordered sets. *Annals of Mathematics* 51(1):161–166.

Dyer, M.; Frieze, A.; and Kannan, R. 1991. A random polynomial-time algorithm for approximating the volume of convex bodies. *Journal of the ACM* 38(1):1–17.

Eiben, E.; Ganian, R.; Kangas, K.; and Ordyniak, S. 2016. Counting linear extensions: Parameterizations by treewidth. In *Proceedings of the 24th Annual European Symposium on Algorithms*, volume 57 of *Leibniz International Proceedings in Informatics*, 39:1–39:18. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

Garey, M. R.; Johnson, D. S.; and Stockmeyer, L. J. 1976. Some simplified NP-complete graph problems. *Theoretical Computer Science* 1(3):237–267.

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2006. Model counting: A new strategy for obtaining good bounds. In *Proceedings of the 21st National Conference on Artificial Intelligence*, 54–61.

Huber, M., and Schott, S. 2010. Using TPA for Bayesian inference. *Bayesian Statistics* 9:257–282.

Huber, M. 2006. Fast perfect sampling from linear extensions. *Discrete Mathematics* 306(4):420–428.

Huber, M. 2014. Near-linear time simulation of linear extensions of a height-2 poset with bounded interaction. *Chicago Journal of Theoretical Computer Science*.

Kangas, K.; Hankala, T.; Niinimäki, T.; and Koivisto, M. 2016. Counting linear extensions of sparse posets. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, 603–609. AAAI Press.

Karzanov, A., and Khachiyan, L. 1991. On the conductance of order Markov chains. *Order* 8(1):7–15.

Kim, C.; Sabharwal, A.; and Ermon, S. 2016. Exact sampling with integer linear programs and random perturbations. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 3248–3254. AAAI Press.

Lukasiewicz, T.; Martinez, M. V.; and Simari, G. I. 2014. Probabilistic preference logic networks. In *Proceedings of the 21st European Conference on Artificial Intelligence*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, 561–566. IOS Press.

Mannila, H., and Meek, C. 2000. Global partial orders from sequential data. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining*, 161–168. ACM.

Möhring, R. H. 1989. Computationally tractable classes of ordered sets. In Rival, I., ed., *Algorithms and Order*, 105–193. Kluwer Academic Publishers.

Morton, J.; Pachter, L.; Shiu, A.; Sturmfels, B.; and Wienand, O. 2009. Convex rank tests and semigraphoids. *SIAM Journal on Discrete Mathematics* 23(3):1117–1134.

Muise, C. J.; Beck, J. C.; and McIlraith, S. A. 2016. Optimal partial-order plan relaxation via MaxSAT. *Journal of Artificial Intelligence Research* 57:113–149.

Niinimäki, T.; Parviainen, P.; and Koivisto, M. 2016. Structure discovery in Bayesian networks by sampling partial orders. *Journal of Machine Learning Research* 17:57:1–57:47.

Peczarski, M. 2004. New results in minimum-comparison sorting. *Algorithmica* 40(2):133–145.

Talvitie, T.; Niinimäki, T.; and Koivisto, M. 2017. The mixing of Markov chains on linear extensions in practice. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. IJCAI.

Wallace, C. S.; Korb, K. B.; and Dai, H. 1996. Causal discovery via MML. In *Proceedings of the 13th International Conference on Machine Learning*, 516–524. Morgan Kaufmann.

Wilson, D. B. 2004. Mixing times of lozenge tiling and card shuffling Markov chains. *The Annals of Applied Probability* 14(1):274–325.